2012

Direct Optimization of Ranking Measures for Learning to Rank Models

Li Li Guo
Wright State University

Direct Optimization of Ranking Measures for Learning to Rank Models

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

by

Li Li Guo
B.S., Wright State University, 2010

2012
Wright State University

Wright State University
SCHOOL OF GRADUATE STUDIES

July 5, 2012

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Li Li Guo ENTITLED Direct Optimization of Ranking Measures for Learning to Rank Models BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Shaojun Wang, Ph.D
Thesis Director

Mateen Rizki, Ph.D
Chair, Department of Computer Science and Engineering

Committee on
Final Examination

Keke Chen, Ph.D

Shaojun Wang, Ph.D

XinHui Zhang, Ph.D

Andrew Hsu, Ph.D
Dean, School of Graduate Studies

ABSTRACT

Guo, Li Li. M.S., Department of Computer Science and Engineering, Wright State University, 2012. *Direct Optimization of Ranking Measures for Learning to Rank Models.*

The main challenge in learning-to-rank for information retrieval is the difficulty to directly optimize ranking measures to automatically construct a ranking model from training data. It is mainly due to the fact that the ranking measures are determined by the order of ranked documents rather than the specific values of ranking model scores, thus they are non-convex, nondifferentiable and discontinuous. To address this issue, listwise approaches have been proposed where loss functions are defined either by exploiting a probabilistic model or by optimizing upper bounds or smoothed approximations of ranking measures. Even though very promising results have been achieved, there is still a mismatch between target cost and optimization cost. In this work, we present a novel learning algorithm that directly optimizes the ranking measures without resorting to any upper bounds or approximations. Our approach is essentially an iterative greedy coordinate descent method in optimization. For each iteration, we only update one parameter along one coordinate with all others fixed. Since the ranking measure is a stepwise function of a single parameter, we exploit an exhaustive line search algorithm to locate the interval with the smallest ranking measure along each coordinate. We pick the coordinate that leads to the largest reduction of ranking measure. In order to determine the optimal value of the parameter for the selected coordinate, we construct a probabilistic framework for the permutation, and maximize the likelihood of top-m ranked documents. This iterative procedure is continued until convergence. We conduct experiments of five datasets selected from Microsoft LETOR datasets, our experimental results show that the proposed direct rank algorithm outperforms several well-known state-of-the-art ranking algorithms.

Contents

List of Figures

List of Tables

Acknowledgement

Foremost, my sincere gratitude goes to my advisor Prof. Shaojun Wang for receiving me to join the lab and for his kindness, motivation, expertise, and most of all, for his patience. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Master study. I would like to thank my lab mates in Machine Learning Group: Ming Tan, Tian Xia and Shaodan Zhai.

Dedicated to

my dear parents, Houmin Guo and Chunlan Ou; my brother, Zhenghe Guo

without whom it would never have been accomplished

Introduction

1.1 Overview

Ranking is a key component of many information retrieval problems, such as document retrieval, collaborative filtering, sentiment analysis, online advertisement placement. Apart from information retrieval, ranking has been applied to computational biology, proteomics [12], recommender system [16], etc. Learning-to-rank, as a supervised machine learning technique, emerges as a new research topic in ranking that aims to automatically build a ranking model from training data. Training data consists of queries and documents that are matched with relevance degree. The relevance degree is usually annotated by human evaluators, who assess the results for some queries and determine the relevance of each result. In testing, the ranking model yields a permutation of items in new and unseen lists, and a series of evaluation standards can be applied to qualify the output.

1.2 Evaluation measures

There are several measures which are commonly used to judge the performance of an algorithm, including MAP (Mean Average Precision), Precision, MRR (Mean Reciprocal Rank) and NDCG (Normalized Discounted Cumulative Gain)[15].

For the query $q \in Q$, rank position of the first relevant document is denoted as

$Rank(q)$, while the documents ranked below $Rank(q)$ are not considered. Mean reciprocal rank is computed as following.

$$MRR = \frac{1}{|Q|}\sum_{i=0}^{|Q|}\frac{1}{Rank(q_i)} \tag{1.1}$$

Average Precision for q_i is defined as:

$$AvgP_i = \frac{\sum_{j=1}^{m(q_i)}\frac{\sum_{k:\pi_i(k)\leq\pi_i(j)}y_{ik}}{\pi_i(j)}y_{ij}}{\sum_{j=1}^{m(q_i)}y_{ij}} \tag{1.2}$$

where $m(q_i)$ is the candidate size retrieved according to q_i, y_{ik} is the human-labeled judgment, and $\pi_i(k)$ is the position of the candidate k within a ranked list of q_i

Another widely used evaluation metric, Normalized Discounted Cumulative Gain, will be discussed in Section 3.1. The most essential feature shared by metrics is that they are a scalar averged among each query, which yields a greater value if the more relavant candidates are ranked higher.

1.3 General learning-to-rank algorithms

Learning-to-rank algorithms can generally be grouped into three categories. The first category is pointwise approach [9, 11, 23]. In this approach, it is assumed that each query-document pair has a numerical or ordinal score. Ranking is formulated as a classification or regression problem, in which the rank value of each document is generally computed independently as an absolute quantity. Methods in this category are slightly different from conventional machine learning algorithms, and cannot handle pairwise preference and orders.

The second category is the pairwise approach [8, 10, 13]. In this approach, the ranked list is decomposed into a set of document pairs. Ranking is treated as a classification prob-

lem that can determine which document is better than the other for document pairs. The goal here is to minimize the number of inversions in ranking. For example, Rankboost [10] plugs the exponential loss of document pairs into a framework of Adaboost[22]; FRank[28] defines a new bounded loss function called fidelity; RankNet[3] defines a logistic loss for document pairs and uses cross entropy as the loss function, and RankSVM[13] uses SVM to perform binary classification on these instances. These approaches outperform the pointwise ones by considering the document's pairwise relationship. However they still ignore the ground truth with respect to partial or total orders of retrieved documents.

The third category is the listwise approach [6, 29, 30, 32]. In this approach, the entire ranked list of documents for each query is taken into account as a training item. A brief discussion of this category is provided in Section 2.1.

1.4 Research motivations

Ideally, the listwise methods should directly optimize the ranking measures. However, direct optimization of ranking measures is challenged by a major difficulty: the ranking measures are non-convex, nondifferentiable and discontinuous due to the fact that the ranking measures are determined by the ranked position of documents rather than an explicit value of each document's ranking score function. Previous studies partially solved this problem by using surrogate functions of ranking measures, nevertheless it is still an open question of how to resolve a mismatch between the objective function used in training and the final evaluation criterion used to measure the task performance.

In this thesis, we present a novel algorithm that is able to train the ranking model by directly optimizing the same ranking measures as used in the testing phase. Our approach is essentially an iterative greedy coordinate descent method in optimization. For each iteration, we only update one parameter along one coordinate with all others fixed. Since the ranking measure is a stepwise function of a single parameter, we exploit an exhaustive

line search algorithm to locate the interval with the smallest ranking measure. We pick the coordinate that leads to the largest reduction of ranking measure. In order to determine the optimal value of the parameter for the selected coordinate, we construct a probabilistic framework for the permutation, and maximize the likelihood of top-m candidates. This iterative procedure is continuned until convergence.

We conduct experiments on five datasets selected from Microsoft LETOR datasets, our experimental results show that the proposed direct rank algorithm outperforms several well-known ranking algorithms.

The rest of the thesis is organized as following; We give a brief summary of related works in Chapter 2. In Chapter 3, we describe our proposed iterative greedy coordinate descent method. Experimental results and discussions are shown in Chapter 4. Chapter 5 concludes this thesis and discusses future work.

Related work

In this Chapter, we first discuss the listwise approaches, and then briefly introduce the MERT algorithm [17] that inspires our work for information retrieval.

2.1 Listwise approaches

Essentially, listwise approaches can be categorized into two groups, with respect to the optimized objective functions. The first group defines a probabilistic framework and optimizes a listwise loss function as an indirect way to optimize the IR evaluation measures. For example, ListNet[6] defines a probabilistic framework for the permutation of retrieved documents and adopts the cross entropy as the optimization objective; RankCosine[20] defines the loss function on the basis of the cosine similarity between the score vector of the ground truth and that of the predicted result; in [30], the authors questions the soundness of using the cross entropy function or cosine similarity and proposes ListMLE[30] which maximizes the likelihood of the ground truth permutations under the same probabilistic framework proposed in [6].

The second group aims to optimize an upper bound or a closely related approximation of ranking measures. For upper bound examples, AdaRank[32] takes an exponential form of the original loss as the upper bound and heuristically introduces a probabilistic distribution over training queries of the training data, and uses a rather ad hoc rule to approximately determine the weight of weak rankers; SVM-MAP [34] minimizes a hinge loss function, an

upper bound of loss function based on Average Precision; in [33], the authors describe two types of upper bounds, and propose PermuRank that is based on the type two bound. some other examples use smoothed functions for approximation and conduct optimization on the surrogate objective functions. NDCGBoost[29] presents a probabilistic framework and optimizes the expectation of NDCG value over all the possible permutations of documents. A relaxation strategy is performed to approximate the average of NDCG over the space of permutation, and a bound optimization approach is used for the efficient computation; SoftRank[27] gives an approximation of NDCG so as to make it smooth and differentiable; and Sun et al.[31] reduce the ranking, as measured by NDCG, to pairwise classification and apply alternating optimization strategy to address the sorting problem by fixing the rank position to get the derivative. [24] investigate the theoritical foundations of the direct optimizing approach, and gives a proof that under certain conditions, no other approach can outperform the direct optimizing metric approaches in the large sample limit. They also propose a general framework, which can accurately approximate any position-based IR measures, and developed ApproxAP and ApproxNDCG for corresponding surrogate functions. In [7], an annealing algorithm is proposed to iteratively minimize a less and less smoothed approximation of the optimized measures. In [4], LambdaRank defined a virtual gradient on each document after the sorting in order to attack the difficulty in optimizing IR metrics. [4] also provided a simple test to determine if there exists an implicit cost function for the virtual gradient. LambdaMART in [5] is the boosted tree version of LambdaRank. A ensemble of LambdaMART rankers won Track 1 of the 2010 Yahoo Learning To Rank Challenge.

Even though all the listwise approaches discussed above target directly optimizing ranking measures, due to the discontinuity of the ranking measures, they have to substitute the ranking measures with surrogate functions as optimization objective functions during training, there is still a mismatch to the test objectives. In comparison, our proposed algorithm is a greedy coordinate descent method that directly optimizes the ranking measure.

Our work is mainly inspired by the minimum error rate (MERT) algorithm [17] that is used in machine translation to re-rank translated candidate sentences. We give a brief introduction in the next section.

2.2 MERT algorithm

In [17], the author shows that better machine translation results can be obtained if the final evaluation criterion is taken into account directly during training. Specifically, assume $E(r, e)$ measures the error between translated sentence e and a reference sentence r. Given a training corpus that contains a set of input sentences $\{f_i\}_{i=1}^{S}$ with given reference translations $\{r_i\}_{i=1}^{S}$, and a set of different candidate translations C_i for each input sentence f_i, MERT tries to find model parameters $\underline{\lambda}^{\star}$ that minimizes total error of translated sentences over the training corpus,

$$\underline{\lambda}^{\star} = \underset{\lambda}{\text{argmin}} \left\{ \sum_{i=1}^{S} E(r_i, \hat{e}_i) \right\} \tag{2.1}$$

with

$$\hat{e}_i = \underset{e_i \in \mathcal{C}_i}{\text{argmax}} \left\{ \sum_{m=1}^{M} \lambda_m h_m(e_i|f_i) \right\} \tag{2.2}$$

$\hat{e}_i$ is the top one ranked translation candidate by a linear function with a fixed set of predefined features $h_m(e_i|f_i), m = 1, \cdots, M$ between translated sentence e and input sentence f, $\underline{\lambda} = (\lambda_m, m = 1, \cdots, M)$ is the feature weight vector that needs to be estimated. Due to the argmax operation in Eqn. (2.2), it is not possible to compute a gradient, thus gradient descent methods cannot be applied to perform optimization.

The general idea of the MERT algorithm is to start from a random started initial value in the parameter space, and use the coordinate descent to optimize the objective function. That is for each iteration, only one parameter is tuned and other parameters are kept fixed.

To address the discontinuity, the author in [17] proposed an efficient algorithm to traverse the error surface, and locate the parameter interval with a minimal error count.

Inspired by the MERT algorithm, we propose the DirectRank algorithm that is able to directly optimize arbitrary nonconvex, nondifferential and discontinuous objective functions determined by top m candidates, such as MAP, NDCG, etc. ranking measures.

In this work, we use NDCG as the ranking measure to illustrate the main ideas in DirectRank, and of course other ranking measures can be straightforwardly plugged-in for practical applications.

DirectRank: A direct method to optimize ranking measures

In this section, we first introduce the notations used in this work and describe the general framework of the proposed DirectRank algorithm. As mentioned above, DirectRank is essentially a greedy coordinate descent method. Therefore, we introduce an efficient exhaustive line search algorithm to identify the best interval of a tuned parameter along one coordinate. Finally, for a selected coordinate that leads to the largest reduction of ranking measure, we propose a probabilistic model to determine the optimal value of the tuned parameter in the best interval by maximizing the likelihood of top-m ranked documents.

3.1 General framework

Suppose that a set of training queries $Q_s = \{q_1, q_2, \cdots q_n\}$ is given, and a set of documents $\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,m(q_i)}\}$ is retrieved for each query q_i. Let $m(q_i)$ denote the size of the set of retrieved documents, which varies for different queries. Every document d_{ij} is associated with a manually-labeled judgment $y_{ij} \in \{r_1, r_2, \cdots, r_l\}$, that denotes the relevance of a document to its belonging query. We define the order $r_l \succ r_{l-1} \succ \cdots \succ r_2 \succ r_1$, where $\succ$ means the preference relationship. A feature vector is created for each document d_{ij} and is denoted as $\mathbf{h}(d_{ij}|q_i) = (h_1(d_{ij}|q_i), h_2(d_{ij}|q_i), \cdots, h_T(d_{ij}|q_i))$. T is the size of

Notation	Explanation
q_i	i^{th} query in the training set. $i \in$ (1,...,n)
d_{ij}	j^{th} document of the query q_i. $j \in$ (1,...,$m(q_i)$)
y_{ij}	the relevance of document d_{ij}.
$\mathbf{h}(d_{ij}\|q_i)$	the feature vector of document d_{ij}.
$h_t(d_{ij}\|q_i)$	the t^{th} feature of document d_{ij}.
$\underline{\alpha}$	the model parameter vector. $\underline{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_T]^T$
pos$(\pi_i, \mathbf{h}(d_{ij}\|q_i), f)$	the position of d_{ij} in permutation π_i
$\mu(\pi_i, j)$	the document at position j given a permutation π_i
$m(q_i)$	the retrieved document size of q_i
m	the truncated ranking size

Table 3.1: Summary of notations

feature vector. π_i is a permutation of the candidates of q_i. We summarize the notations used here in Table 3.1.

The objective of ranking is to construct a ranking function $f : R^T \rightarrow R$, in which the input is the feature vector $\mathbf{h}(d_{ij}|q_i)$ for a document, and the output is a ranking score. Specifically in this work, we use the linear combination of features as our ranking function

$$f(\mathbf{h}(d_{ij}|q_i)) = \alpha_1 h_1(d_{ij}|q_i) + \cdots + \alpha_T h_T(d_{ij}|q_i) \tag{3.1}$$

where the weight vector $\underline{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_T)$ is the model parameter. A variety of ranking measures can be used to assess the performance of the ranking result. Specifically in this thesis, given a training query q_i and the ranking function f, we use the NDCG as the ranking measure, its value is computed as following: Define $\mathcal{G}(d_{ij}, f)$ for each query-document pair and their normalized sum $\mathcal{N}(q_i, f)$ for each query.

$$\mathcal{G}(d_{ij}, f) = \frac{2^{y_{ij}} - 1}{\log(1 + \text{pos}(\pi_i, \mathbf{h}(d_{ij}|q_i), f))} \tag{3.2}$$

$$\mathcal{N}(q_i, f) = \frac{1}{Z_i} \sum_{j=1}^{m(q_i)} \mathcal{G}(d_{ij}, f) \tag{3.3}$$

where Z_i is a normalization factor to guarantee $\mathcal{N}(q_i, f) \in [0, 1]$. The $NDCG$ value given a training query set Q_s is denoted by $\mathcal{NDCG}(Q_s, f) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(q_i, f)$. Usually, $NDCG$ is trimmed at a certain ranking level m (this means we just consider the top m documents, and ignore the rest of the retrieved documents), so we can change $m(q_i)$ in Equation (5) to a constant value m. Our goal is to search for a model parameter vector $\underline{\alpha}^\star$ that achieves maximum $\mathcal{NDCG}(Q_s, f)$, i.e.,

$$\begin{aligned} \underline{\alpha}^\star &= \arg\max_{\underline{\alpha}} \mathcal{NDCG}(Q_s, f) \\ &= \arg\max_{\underline{\alpha}} \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{Z_i} \sum_{d_{ij} \in TOP_m(q_i)} \mathcal{G}(d_{ij}, f) \right) \end{aligned}$$

where,

$$TOP_m(q_i) = \arg\max_{top_m} f(\mathbf{h}(d_{ij}|q_i)) \tag{3.4}$$

The objective function above is difficult to handle for two reasons: First, it is discontinuous and non-convex with respect to $\underline{\alpha}$, thus we cannot directly use gradient descent algorithms to optimize. Second, the objective have many local optima.

Instead of opitimizing the objective along multiple coordinates, we resort to an iterative greedy coordinate descent method. For each iteration, there will be only one coordinate parameter that is updated, while others are kept unchanged. The rationality of this idea is

Algorithm 1 Greedy coordinate descent algorithm for direct optimization

Require: $Q_s = \{q_i\}_{i=1}^n$

for each parameter α_k **do**
  for $q_i \in Q_s$ **do**
    Do Algorithm 2 for q_i
    // Calculate the NDCG jumping points for q_i
  end for
  Merge all the jumping points and sort them.
  Calculate NDCG between the jumping points.
end for
Get optimal α_k and the interval $[L_{\alpha_k}, R_{\alpha_k}]$ that maximizes NDCG.
Pick the coordinates that lead to largest NDCG increments.
Perform likelihood maximization of top-m candidates to update the parameter.
Respeat 1-11 until convergence.

that the ranking function can be written as a linear function with a single parameter,

$$\begin{aligned} f(\mathbf{h}(d_{ij}|q_i)) &= \sum_{t=1}^{T} \alpha_t h_t(d_{ij}|q_i) \\ &= \alpha_k \cdot h_k(d_{ij}|q_i) + \sum_{t \neq k}^{T} \alpha_t h_t(d_{ij}|q_i) \end{aligned}$$

Since $h_k(d_{ij}|q_i)$ is constant with respect to α_k, and so is the second term, we can re-write these two quantities as a_{ij} and b_{ij}, and convert the equation above to,

$$f(\mathbf{h}(d_{ij}|q_i)) = a_{ij} \cdot \alpha_k + b_{ij} \tag{3.5}$$

Note that for each document retrieved by each query, there is a linear function of α_k. Given an input of α_k, each document will get an output score from this linear function. The order of such scores actually reflects the order of the documents which further determines the NDCG value.

This is illustrated in Figure 3.1, where each of the lines represents a scoring function for a document. At any point of α_k, the rank of the linear function output scores is equiva-

lent to the rank of the documents. Note that a little change of α_k cannot lead to a jump of NDCG value, unless it changes the order of the top-m documents. Obviously, such change in order happens only at the point where two lines intersect. We denote these points as *jumping points*. In Figure 3.1, we draw ten lines corresponding to ten documents, which belong to two queries. Intersections $(p_1, p_2, \cdots p_{12})$ are jumping points.

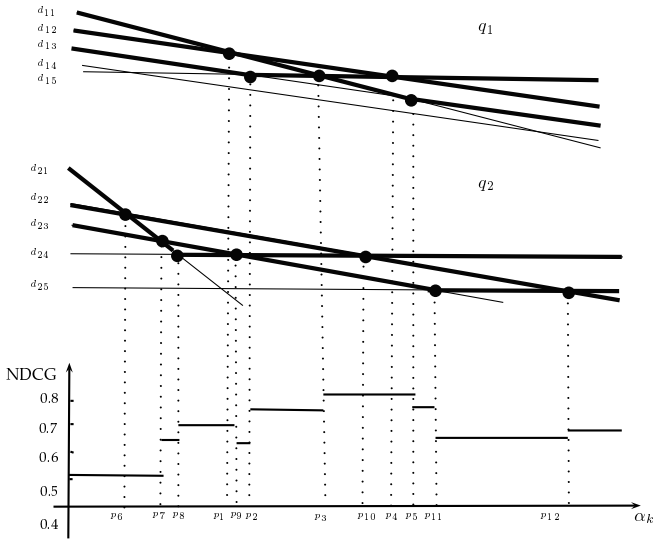

Figure 3.1: Line search algorithm
The top-m candidates for each of the two queries are marked as bold. Between each two boundaries, the NDCG value is shown. We can see the intervals between p_3 and p_5 achieve the best NDCG. The specific values are shown in Table 3.4.

Theoretically, we can brutally search all the intersections to acquire all possible snapshots of ranked documents, Because any two of the non-parallel lines will form an intersection, the total number of intersections then is $m(q_i) \cdot m(q_i)$. This effort is time-consuming but unnecessary, because we are merely interested in the rank of top-m candidates. Thus, the NDCG metric is always truncated to a certain level m, and usually $m \leq$10. The size of jumping points that lead to a change in the top-m candidates are empirically $m \cdot c$, where c is the size of jumping points on the envelope described in [17], and usually less than 10 according to our experiments. This allows us to efficiently find all the jumping points on one coordinate.

The entire procedure of greedy coordinate descent algorithm is described in Algorithm 1. For each coordinate, we exploit such a line search algorithm as follows: For each query, we obtain all of its jumping points (Line 2 $\sim$ 5 in Algorithm 1). For instance, in Figure 2.1, $(p_1, p_2, \cdots, p_5)$ are the jumping points of q_1, while $(p_6, p_7, \cdots, p_{12})$ are the jumping points of q_2. Next, the jumping points of all queries are merged and sorted (Line 6). Then, NDCG can be exactly computed, because of the fact that the objective is a stepwise function. Between any two adjacent jumping points, the top-m candidates of any query stay unchanged, so does the NDCG value (For example, in Figure 3.1, the NDCG will not change between p_3 and p_{10}). Therefore, we exploit a line search algorithm on each coordinate direction and finally pick the parameter and the interval corresponding to the best ranking measure.

In Section 3.3, we introduce the line search algorithm. Given a query, we can precisely find all the jumping points. Moreover, our experiments show that there might be multiple intervals that achieve the best ranking measure. Choosing which interval and which point within the interval severely influences the performance. This is distinguished from the MERT algorithm for machine translation. Thus in section 2.4, we place our focus within the intervals which achieve the optimal NDCG value. We propose to build a probabilistic model on permutations and determine the optimal value of model parameter by maximizing the likelihood of top-m candidates.

3.2 Searching for jumping points

We again use Figure 3.1 to illustrate the line search algorithm. Suppose there are only two queries in the training set, q_1 and q_2, each of which consists of five documents that are denoted as $\mathbf{d_1} = \{d_{11}, d_{12}, \cdots, d_{15}\}$ and $\mathbf{d_2} = \{d_{21}, d_{22}, \cdots, d_{25}\}$ respectively. We assign their relevance judgments as $\{1, 2, 0, 1, 2\}$ and $\{0, 0, 2, 1, 1\}$. Suppose we intend to maximize $NDCG@3$, and obtain the $TOP_3(q_i)$ on the direction of the k^{th} coordinate. This means we apply coordinate descent method and search optimal value of the k^{th} parameter

along the k^{th} coordinate, while keeping all other parameters fixed.

Interval	$< p_1$	$> p_1$	$> p_2$	$> p_3$	$> p_4$	$> p_5$
Rank	d_{11}	$\mathbf{d_{12}}$	d_{12}	d_{12}	$\mathbf{d_{15}}$	d_{15}
	d_{12}	$\mathbf{d_{11}}$	d_{11}	$\mathbf{d_{15}}$	$\mathbf{d_{12}}$	d_{12}
	d_{13}	d_{13}	$\mathbf{d_{15}}$	$\mathbf{d_{11}}$	d_{11}	$\mathbf{d_{13}}$
	d_{14}	d_{14}	d_{14}	d_{14}	d_{14}	d_{14}
	d_{15}	d_{15}	$\mathbf{d_{13}}$	d_{13}	d_{13}	$\mathbf{d_{11}}$

Table 3.2: Ranked documents of q_1 along α_k.
$< p_1$ means the left side of p_1, while $> p_1$ means the right side of p_1, but on the left side of the next point p_2. Bold items denote any changes on the order of ranked documents when sweeping from left to right.

Interval	$< p_6$	$> p_6$	$> p_7$	$> p_8$	$> p_9$	$> p_{10}$	$> p_{11}$	$> p_{12}$
Rank	d_{21}	$\mathbf{d_{22}}$	d_{22}	d_{22}	d_{22}	$\mathbf{d_{24}}$	d_{24}	d_{24}
	d_{22}	$\mathbf{d_{21}}$	$\mathbf{d_{23}}$	d_{23}	$\mathbf{d_{24}}$	$\mathbf{d_{22}}$	d_{22}	$\mathbf{d_{25}}$
	d_{23}	d_{23}	$\mathbf{d_{21}}$	$\mathbf{d_{24}}$	$\mathbf{d_{23}}$	d_{23}	$\mathbf{d_{25}}$	$\mathbf{d_{22}}$
	d_{24}	d_{24}	d_{24}	$\mathbf{d_{21}}$	d_{21}	d_{21}	d_{21}	d_{21}
	d_{25}	d_{25}	d_{25}	d_{25}	d_{25}	d_{25}	$\mathbf{d_{23}}$	d_{23}

Table 3.3: Ranked documents of q_2 along α_k.
$< p_6$ means the left side of p_6, while $> p_6$means the right side of p_6, but to the left side of the next point p_7. Bold items denote any changes on the order of ranked documents when sweeping from left to right.

As shown in Figure 3.1, $(p_1, p_2, \cdots, p_5)$ are the jumping points of q_1, while $(p_6, p_7, \cdots, p_{12})$ are the jumping points of q_2. Table 3.2 and Table 3.3 respectively show the rank of candidates between those jumping points, and the changes of rank are noted as bold. Clearly, a change of document rank order happens when two linear functions of documents intersect at a jumping point. For example, at p_1, d_{11} and d_{12} intersect, which causes a switch between d_{11} and d_{12} in Table 3.2.

Algorithm 2 shows how to precisely compute these points. First, all the lines belonging to the same query are sorted by a_{ij}. This order is actually the rank when α_k takes negative infinity (Line 2 $\sim$ 3 in Algorithm 2). For the example of q_1, the initial rank is $d_{11}, d_{12}, d_{13}, d_{14}, d_{15}$. Next, we repeat the following procedure in order to find the next intersection that leads to a change of top-m candidates (Line 4 $\sim$ 22). Since the top-m candidates have already been sorted, for each of the top m-1 candidates, we just calculate their intersection with the candidate right below it (Line 4 $\sim$ 8). For example, we calculate

Intervals	$< p_6$	$> p_6$	$> p_7$	$> p_8$	$> p_1$	$> p_9$	$> p_2$
NDCG	0.506	0.506	0.626	0.694	0.694	0.614	0.757
Intervals	$\mathbf{> p_3}$	$\mathbf{> p_{10}}$	$\mathbf{> p_4}$	$> p_5$	$> p_{11}$	$> p_{12}$	
NDCG	**0.812**	**0.812**	**0.812**	0.765	0.629	0.669	

Table 3.4: $NDCG@m, m = 3$ values along the direction of α_k.
$< p_6$ means the left side of p_6, while $> p_6$ means the right side of p_6, but to the left side of the next point p_7. Bold items show the intervals with optimal $NDCG@m, m = 3$ values.

the interection between d_{11} and d_{12}, the one between d_{12} and d_{13}. On the other hand, for the candidate at the rank of m, all the candidates below it have to be scanned, because they might not be sorted, which means any of these candidates might be just below the m^{th} candidate (Line 9 $\sim$ 13). That means for d_{13}, we have to calculate its interections with d_{14} and d_{15}, respectively. We choose the minimum one from these intersections, which is the next jumping point. We update the rank according to the two intersected candidates on this jumping point (Line 16 $\sim$ 20). Such repeated procedure does not terminate until all the jumping points are obtained. For example, as for q_1 in Figure 3.1, its jumping points will generate from left to right as a sequence: $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \rightarrow p_5$.

After that, the jumping points of q_1 and q_2 are merged and sorted, and the NDCG between them can be straightforwardly calculated. The stepwise NDCG values are shown both at the bottom of Figure 3.1 and in Table 3.4. Note that when calculating the NDCG values between the jumping points, it is completely unnecessary to re-calculate the entire training queries. Instead we can update NDCG values in an incremental manner. Since at each jumping point there are only two candidates in one query switching their rank order, therefore we just need to update the NDCG value for the corresponding query.

Finally, we can give an analysis on the complexity of the proposed line search algorithm. The searching procedure of the next intersection on the mth candidate is just as same as the envelope detection introduced in [17], since the candidates ranked below the mth candidate are not sorted and we have to calculate the intersection between the mth candidate and all candidates from the position from $m+1$ to $m(q_i)$. Besides, for the candidates from the position from 1 to $m-1$, since they interects only with the adjacent candidates,

the times of calculating their next intersection between them is $(m-1)$. Therefore, every time we generate the next jump point, we calculate $m(q_i)-1$ times of interections. This is the same as Och's MERT algorithm. However, as discussed in previous section, we have $m\dot{c}$ jumping points, where c is the points of the envelope in [17]. Thus, the calculation of interections is at most $m \cdot c \cdot (m(q_i)-1)$.

Algorithm 2 Line search algorithm

Require: q_i, $\mathbf{d_i} = \{d_{ij}\}_{j=1}^{m(q_i)}$

1: Compute the linear function $a_{ij} \cdot \alpha_k + b_{ij}$ for each candidate
2: Sort the candidates by a_{ij}, and get an initial rank of candidates R
3: CurrP $\leftarrow$ -inf
4: $M \leftarrow 1$
5: **while** $M < m$ **do**
6: Calculate the intersection of $R[M]$ and $R[M+1]$
7: $M \leftarrow M+1$
8: **end while**
9: $M \leftarrow m+1$
10: **while** $M \leq m(q_i)$ **do**
11: Calculate the intersection of $R[m]$ and $R[M]$
12: $M \leftarrow M+1$
13: **end while**
14: Get the minimum intersection I, such that $I>$CurrP
15: **if** I is not empty **then**
16: $R[U_I] \leftrightarrow R[D_I]$
17: // I is a jumping point.
18: // U_I and D_I intersect at point I
19: CurrP $\leftarrow I$
20: Go to Line 4
21: **else**
22: Finish
23: **end if**

In the next section, we focus on determining the optimal value of the model parameter within the intervals which achieve the optimal NDCG value.

3.3 Determining the optimal value of model parameters

The original Och's MERT algorithm finishes for each iteration by finding the interval with minimal error rate. The value of model parameter is determined either by a random point in the interval or the middle point of the interval. In this section, we propose a strategy to determine the optimal value of the model parameter in the chosen interval. We find that this is critical for the success of DirectRank since if we choose a random point in the interval or the middle point of the interval, we obtain worse results. For such a purpose, we build an appropriate probabilistic model for the top-m candidates.

Many probabilistic models have been proposed previously, such as the Bradley-Terry-Luce model [2], the Mallows model [18], and the Plackett-Luce model [2, 18]. The Plackett-Luce model was successfully adopted by [6, 14]. It treats the ranking procedure as a random selection sequence without placement. In this work, we adopt the form of the Plackett-Luce model in our DirectRank algorithm, and define the permutation probability as,

$$p(\pi_i; q_i, f) = \prod_{j=1}^{m(q_i)} \frac{\exp(f(d_{i,\mu(\pi_i,j)}))}{\sum_{l=j}^{m(q_i)} \exp(f(d_{i,\mu(\pi_i,l)}))} \tag{3.6}$$

where j and k are the rank indices and $\mu(\pi, j)$ denotes the document of the position j in a permutation π. In [6], the authors clarified an important property for this form of the Plackett- Luce Model: Given a scoring function, the ranked list of the documents sorted based on the scores has the highest permutation probability, while the list of documents sorted in the inverse order has the lowest permutation probability. The property implies that choosing the ranked list with the highest probability is equivalent to the way a typical ranking function selects a list.

Same as in [14], the probability of the top-m ranked documents can be similarly de-

fined as,

$$p(\pi_1^m; q_i, f) = \prod_{j=1}^{m} \frac{\exp(f(d_{i,\mu(\pi_i,j)}))}{\sum_{l=j}^{m(q_i)} \exp(f(d_{i,\mu(\pi_i,l)}))} \tag{3.7}$$

Here the scoring function f is linear, and the only parameter estimated is α_k , while others in $\underline{\alpha}$ are fixed. The log probability of top-m ranked documents for all training queries can be formulated as,

$$\begin{aligned} \log P(Q_s^m; f) &= \sum_{i=1}^{n}\sum_{j=1}^{m} \log \frac{\exp(f(d_{i,\mu(\pi_i,j)}))}{\sum_{l=j}^{m(q_i)} \exp(f(d_{i,\mu(\pi_i,l)}))} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} \left(a_{i,\mu(\pi_i,j)} \cdot \alpha_k + b_{i,\mu(\pi,j)}\right) \\ &\quad -\sum_{i=1}^{n}\sum_{j=1}^{m} \left(\log \sum_{l=j}^{m(q_i)} \exp(a_{i,\mu(\pi,l)} \cdot \alpha_k + b_{i,\mu(\pi,l)}) \right) \end{aligned} \tag{3.8}$$

It's easy to prove that the likelihood loss is continuous, differentiable, and convex [1]. This is one dimensional concave optimization problem, and we maximize the log likelihood by the binary search method. The new of α_k must not exceed the interval with maximum NDCG value, located by Algorithm 2 stated in the previous section.

Experimental results

4.1 Introduction

We conduct extensive experiments to test the performance of DirectRank algorithm on five datasets on LETOR 3.0 and LETOR 4.0. LETOR is a package of benchmark data sets for research on learning to rank, and it contains standard features, relevance judgments, data partitioning and evaluation tools as well as several baseline results of the state-of-the-art ranking algorithms [19]. LETOR datasets prevent us from the process of sample gathering and feature selection, and enable us to focus our emphasis just on the performance of learning-to-rank algorithms themselves.

4.2 Experimental Set-up

In LETOR 3.0, we select OHSUMED data, and two .gov datasets, that include homepage finding 2003 (HP2003), and topic distillation 2004 (TD2004). In LETOR 4.0, we use Million Query track of TREC 2007 and 2008 (MQ2007 and MQ2008). For each of these datasets, five fold partitions are generated for cross validation, including one training, one testing and one validation data. All experimental results reported below are those averaged on these five fold cross validation datasets.

We compare DirectRank with four well-known state-of-the-art learning to rank algo-

rithms: AdaRank-NDCG[32], RankSVM[13] and RankBoost[10] and FRank[28]. AdaRank-NDCG is a listwise boosting algorithm that incorporates NDCG in computing the weighted distribution over queries and feature combination weights, the other three are pairwise methods.

The training and test procedures are conducted under the same truncated rank levels m. For example, in training, we estimate the feature weights by maximizing the $NDCG@4$ value of the training queries, then during the test, we calculate the $NDCG@4$ of the test data. The value of m varies from 1, 2, 4, 5, 9 to 10.

Since initial values of feature weights will lead to different results, we repeat the training and test procedure by randomly selecting the initial weights for five times. And we use the model which has the maximum NDCG value in training for testing. We also make use of validation datasets, in the case that when two trials above have the same training NDCG value, we choose the one with the higher NDCG value on validation datasets. The training procedure is terminated when for any two successive iterations of the greedy coordinate descent method, the difference of model parameters is within a predefined small threshold.

One thing we need to acknowledge is that we failed to reproduce a large part of results released on LETOR website, partially for the reason that the LETOR datasets did not release adequate details about experimental configuration for each baseline approaches. For example, we tried to get the released numbers of the nine datasets of LETOR 3.0 and 4.0, with the AdaRank-NDCG execurable published by Microsoft. It indicate some significant difference between the published numbers and our results. However, due to the limit of time, the baseline results are directly referred to the ones released from the LETOR dataset.

$NDCG@m$	1	2	4	5	9	10
Rankboost	0.463	0.450	0.454	0.449	0.432	0.430
FRank	0.530	**0.500**	0.468	0.458	0.446	0.443
RankSVM	0.495	0.433	0.424	0.416	0.412	0.414
AdaRank	0.533	0.492	0.468	0.467	0.454	0.449
DirectRank	**0.564**	0.496	**0.472**	**0.473**	**0.458**	**0.455**

Table 4.1: The ranking accuracies on OSHUMED data

4.3 Experiments on LETOR 3.0 Data

In the first experiment on LETOR 3.0 data, we use the OHSUMED dataset to test the performance of DirectRank. In OSHUMED dataset, there are 16,140 query-document pairs upon which relevance judgments are given, and the total number of queries is 106. The relevance judgments are "definitely relevant", "possibly relevant" or "not relevant", which are represented by the preference level of 2, 1, and 0 respectively. As shown in Table 4.1, although FRank outperforms other algorithms on $NDCG@2$, the overall best baseline is produced by AdaRank. Our DirectRank algorithm achieves significant improvements on all presented measures from 0.45% to 3.1%, compared to AdaRank.

In the second experiment on LETOR 3.0 data, we use the TREC .Gov data to test the performance of DirectRank for the task of web retrieval. Specifically, we show that the ranking results on homepage finding 2003 (HP2003) dataset, and topic distillation 2004 (TD2004) dataset, whose 64 features are already provided in LETOR 3.0 dataset. Both HP2003 and TD2004 datasets contains 75 queries, each of which has about 1000 retrieved queries. Table 4.2 shows the $NDCG@m$ results of HP2003 datasets by DirectRank algorithm and the other four baseline algorithms. Among the four baseline algorithms, RankBoost generally yields the best results, except the case of $\mathrm{m} = 1$. And DirectRank works better than all cases than RankBoost algorithm, and the improvement is up to 1.1%.

In Table 4.3, the $NDCG$ values are given on the TD2004 datasets. RankBoost performs the best among all the presented baseline algorithms, while our proposed method is considered to be very competitive with the best results.

$NDCG@m$	1	2	4	5	9	10
Rankboost	0.666	0.770	0.792	0.803	0.816	0.817
FRank	0.653	0.730	0.763	0.778	0.790	0.797
RankSVM	0.693	0.746	0.787	0.795	0.807	0.807
AdaRank	**0.713**	0.766	0.794	0.800	0.805	0.806
DirectRank	0.666	**0.778**	**0.798**	**0.809**	**0.827**	**0.822**

Table 4.2: The ranking accuracies on HP2003 data

$NDCG@m$	1	2	4	5	9	10
Rankboost	0.506	0.433	**0.405**	**0.387**	**0.352**	0.312
FRank	0.492	0.406	0.358	0.362	0.333	0.269
RankSVM	0.413	0.346	0.341	0.324	0.306	0.307
AdaRank	0.426	0.380	0.352	0.351	0.321	0.316
DirectRank	**0.508**	**0.433**	0.399	0.383	0.347	**0.322**

Table 4.3: The ranking accuracies on TD2004 data

4.4 Experiments on LETOR 4.0 Data

In the first experiment on LETOR 4.0 Data, we use two query sets from Million Query track of TREC 2007 and TREC 2008 in LETOR 4.0 Data, i.e., MQ2007 and MQ2008. In these datasets, each query-document pair contains 46 features. Since LETOR 4.0 dataset does not provide published results run by FRank, here we only compare DirectRank with the other three baseline ranking algorithms.

For MQ2007 dataset, Table 4.4 shows that except $NDCG@1$, the DirectRank algorithm significantly outperforms RankBoost, the second best ranking algorithm in this experiment, and the improvement can be as much as 0.7%.

Table 4.5 shows the $NDCG$ results on MQ2008 dataset. Apparently DirectRank obtains the best results than the baseline ranking algorithms. Especially, in the cases that the

$NDCG@m$	1	2	4	5	9	10
Rankboost	**0.413**	0.409	0.412	0.418	0.439	0.446
RankSVM	0.409	0.407	0.408	0.414	0.436	0.443
AdaRank	0.387	0.396	0.406	0.410	0.431	0.436
DirectRank	0.407	**0.413**	**0.413**	**0.418**	**0.443**	**0.453**

Table 4.4: The ranking accuracies on MQ2007 data

$NDCG@m$	1	2	4	5	9	10
Rankboost	0.385	0.399	0.447	0.466	0.221	0.225
RankSVM	0.362	0.398	0.450	0.469	0.223	0.227
AdaRank	0.382	0.421	0.465	0.482	0.227	0.230
DirectRank	**0.392**	**0.426**	**0.470**	**0.484**	**0.513**	**0.483**

Table 4.5: The ranking accuracies on MQ2008 data

truncated level is increased to 9 and 10, the $NDCG$ values of three baseline ranking algorithms drop down drastically, however, this phenomena does not happen for our proposed DirectRank. Therefore DirectRank has over 25 percent of $NDCG$ increments over three baseline ranking algorithms.

In summary, among the baseline ranking methods over all of the datasets, AdaRank gives the best performance on OSHUMED and MQ2008 datasets, whereas RankBoost outperforms other baseline ranking algorithms on HP2003, TD2004 and MQ2007 datasets. However, none of the baseline ranking methods achieves consistent best performance over all of the datasets. In contrast, our proposed DirectRank method outperforms the best baseline ranking algorithms over all of the datasets with respect to most of $NDCG@m$ measurements.

4.5 Discussion

In this section, we give some detailed analysis on our proposed DirectRank algorithm using the OSHUMED data as an example. We compare DirectRank with AdaRank[32]. As declared by Xu and Li [32], AdaRank minimizes the exponential loss of NDCG value, defined below,

$$\frac{1}{n}\sum_{i=1}^{n}\exp\left[-\mathcal{N}(q_i, f)\right] \tag{4.1}$$

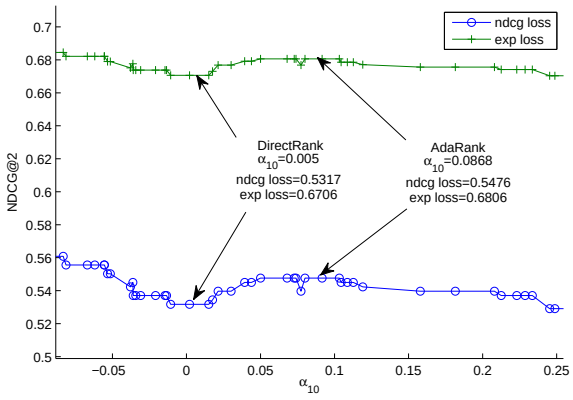

Figure 4.1: NDCG Loss vs. exponential Loss
. There are two kinds of loss changes along α_{10} in the iteration 5. DirectRank and AdaRank choose different optimal points respectively, while DirectRank achieves both minimal exponential loss and minimal NDCG loss.

This is an upper bound of NDCG loss, denoted as,

$$1 - \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(q_i, f) \tag{4.2}$$

As shown in Figure 4.1, the exponential loss and NDCG loss always change consistently, the property of discontinuity and non-convexity still remains in the exponential loss. AdaRank embeds NDCG into AdaBoost. As described in [32], AdaRank adds a new weak ranker h_t with α_k

$$\alpha_k = \frac{1}{2} \cdot \log \frac{\sum_{i=1}^{n} P_t(i)\,[1 + \mathcal{N}(q_i, h_t)]}{\sum_{i=1}^{n} P_t(i)\,[1 - \mathcal{N}(q_i, h_t)]} \tag{4.3}$$

as the weight AdaRank chooses by a rather ad hoc rule on iteration t, and $P_t(i)$ is the weight of query i. Now we show that AdaRank leads to a suboptimal estimation of α_k when minimizing the exponential loss of of NDCG loss. To do this, we just replace the NDCG evaluation metric in this thesis with its exponential loss defined in Eqn. (4.1). Figure 2 shows the 5^{th} iteration of AdaRank while the training set is OSHUMED data and $m = 2$.

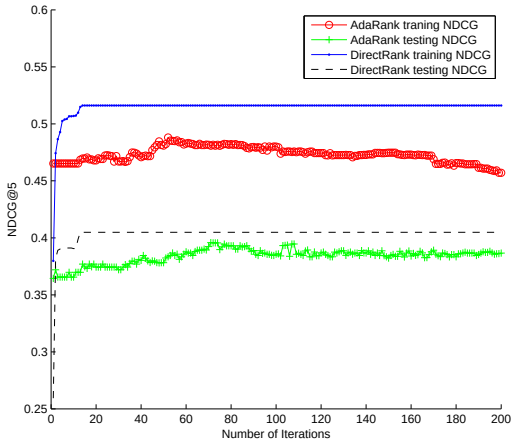

Figure 4.2: The NDCG of AdaRank and DirectRank of different iterations.

AdaRank starts with the parameter $\alpha = [0, 0, \cdots, 0]^T$. After four iterations, $\alpha_7 = 0.343746$ and $\alpha_{11} = 0.097015$, while others remain zero. AdaRank chooses the weak ranker α_{10} by the approach in Section 3.6 in [32]. Therefore, we plot the exponential loss and NDCG loss defined on $NDCG@2$ the same feature in Figure 4.1, while copying the weights generated by AdaRank in first four iterations, We can see that the value of α_{10} located by AdaRank is very different from our DirectRank algorithm, and DirectRank can achieve the minimal exponential NDCG loss. That explains why DirectRank outperforms AdaRank.

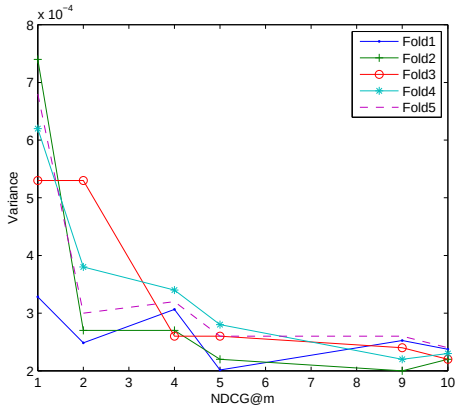

Figure 4.3: The variances of testing NDCG with different random initial parameters. The results of five cross-validation datasets on OSHUMED data are shown.

Figure 4.2 draws the learning curves of DirectRank and AdaRank on OSHUMED data where the NDCG is truncated at level $m = 5$. In order to make a fair comparison, we launch the training process with all initial parameters equal to zero, as same as AdaRank. We can see that the training NDCG of DirectRank constantly increases. And it can get the convergence within 20 iterations and achieves a better performance compared to AdaRank.

Figure 4.3 shows the variances of the NDCG on testing data with five times of different initial parameters. From the figure, we can see that the training performance is influenced by the initial values, especially in the case of $m = 1$. However, performing DirectRank for multiple times with different initial parameters can effectively solve this problem.

Conclusions and future work

In this thesis, we propose a greedy coordinate descent algorithm, DirectRank, for learning ranking models for information retrieval. In contrast to existing listwise methods, DirectRank directly optimize ranking measures instead of their upper bounds or approximations. It uses an efficient exhaustive line search algorithm to identify the interval that has the optimal non-smooth ranking measures along each coordinate, then greedily pick the coordinate leading to the largest reduction ranking measure, and finally update the parameter along this coordinate which has the maximum likelihood value. DirectRank offers several advantages: ease of implementation, efficiency in training, and high accuracy in ranking. Experimental results based on five benchmark datasets published in LETOR 3.0 and 4.0 show that, in most cases, DirectRank significantly outperforms the state-of-the-art baseline ranking methods such as RankSVM, RankBoost, FRank and AdaRank. It is the first method that is able to directly optimize ranking measures without any approximation.

Our research can be extented on the following aspects. Firstly, instead of one-dimensional coordinate descent, we can consider a coordinate transformation, targeting to optimize multiple parameters in a similar manner at one time. Secondly, we will extend DirectRank into a semi-supervised version, such that we are able to make use of a huge amount of unlabeled query-document pairs in the real practice. The challenge include how to figure out an appropriate objective function with respect to the query-document pairs with no human-labeled relevance judgments. Thirdly, we would adapt the margin theory of SVM into our algorithm, in which we intend to maximize the margin between the top-m candidates which

are used to calculate ranking measures and other candidates.

Moreover, we plan to conduct a series of experiments on different datasets to verify our proposed method and to draw convincing conclusion. Besides of other small-scale LETOR datasets, we would test our algorithm on larger datasets, such as the Yahoo Learning-to-rank Challenge Datasets. In order to futher verify our approach, we plan to evaluate it on datasets with much larger amount of features. The complexity of our algorithm grows linearly with the increase of features, therefore, it will be feasible computation load to train Yahoo challenge datasets as long as a distributed version of our new algorithm is developed. To achieve this, our research group has an easy access to the resources in Ohio Supercomputer Center, which includes more than 9,000 computational nodes. Moreover, we have already accumulated adequate experience on developing a large-scale distributed software. Our previous research on large-scale composite language model for machine translation makes a world record in Perplexity reduction on N-gram baseline [25, 26].

As far as we know, our algorithm is the first method that is able to directly optimize ranking measures, thus there is no mismatch between training objective and testing objective. Therefore, we expect that our proposed algorithm can outperform other state-of-the-art learning-to-rank algorithms. From the long-term perspective, we are confident that the proposed method can result in a substantially better performance in a variety of ranking applications.

Bibliography

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University, 2004.

[2] R. Bradley and M. Terry. Rank analysis of incomplete block designs. *Biometrika*, 39(3/4):324-345, 1952.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and Greg. Hullender. Learning to rank using gradient descent. *In Proceedings of the 22nd international conference on Machine learning (ICML)*, pages 89-96, 2005.

[4] C. Burges, R. Ragno, and Q. Le. Learning to Rank with Nonsmooth Cost Functions. *Advances in Neural Information Processing Systems 19*, pages 193-200, 2007.

[5] C. Burges. From RankNet to LambdaRank to LambdaMART:An Overview *Microsoft Research Technical Report MSR-TR-2010-82*, 2010.

[6] Z. Cao, T. Qin, T.-Y. Liu, M.-F Tsai and H. Li. Learning to rank: From pairwise to listwise approach. *In Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 129-136, 2007.

[7] O. Chapelle and M. Wu Gradient descent optimization of smoothed information retrieval metrics *Information Retrieval*, 13:216-235, 2010

[8] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243-270, 1999

[9] K. Crammer and Y. Singer. Pranking with ranking. *Advances in Neural Information Processing Systems*, pages 641-647, 2001.

[10] Y. Freund, R. Iyer, R. E. Schapire and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933-969, 2003.

[11] E. F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. *In Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 250-257, 2003.

[12] C. Henneges, G. Hinselmann, S. Jung, et al. Ranking methods for the prediction of frequent top scoring peptides from proteomics data. *Journal of Proteomics and Bioinformatics*, 2:226-235, 2009.

[13] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. *Int. Conf. on Artificial Neural Networks*, Vol. 1, pages 97-102, 1999.

[14] J. Kuo, P. Cheng, and H. Wang. Learning to rank from bayesian decision inference. *In Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 827-836, 2009.

[15] T.-Y. Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[16] Y.H. Lv, T. Moon, P. Kolari, Z.H. Zheng, X.H. Wang, and Y. Chang. Learning to model relatedness for news recommendation. In Proceedings of the International Conference on World Wide Web (WWW), 2011.

[17] F. J. Och. Minimum error rate training in statistical machine translation. *In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160-167, 2003.

[18] R. Plackett. The analysis of permutations. *Applied Statistics*, 24(2):193-202, 1975.

[19] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval Journal*, 13(4):346-374, 2010.

[20] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information processing and management*, 44(2):838-855, 2005.

[21] P. Ravikumar, A. Tewari, and E. Yang. On NDCG consistency of listwise ranking methods. *In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics,volume 15 of JMLR Workshop and Conference Proceedings,* pages 618-626, 2011.

[22] R. E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms.* MIT Press. 2012.

[23] A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. *In Advances in Neural Information Processing Systems*, pages 937-944, 2003.

[24] T. Qin and T.-Y. Liu and H. Li. A general approximation framework for direct optimization of information retrieval measures *Information Retrieval*, 13:375-397, 2010

[25] M. Tan, W. Zhou, L. Zheng, and S. Wang. A Large Scale Distributed Syntactic, Semantic and Lexical Language Model for Machine Translation *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL)*, pages 201-210, 2011.

[26] M. Tan, W. Zhou, L. Zheng, and S. Wang. A Scalable Distributed Syntactic, Semantic and Lexical Language Model *Computational Linguistics*, 2012.

[27] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: Optimizing non-smooth rank metrics. *In Proceedings of the international conference on Web Search and web Data Mining (WSDM)*, pages 77-86, 2008.

[28] M.-F. Tsai, T.-Y. Liu, Q. Tao, H.-H. Chen, W.-Y. Ma. Frank: A ranking method with fidelity loss. *In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383-390, 2007.

[29] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing NDCG measure. *Advances in Neural Information Processing Systems*, pages 1883-1891, 2009.

[30] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. *In Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1192-1199, 2008.

[31] Z. Sun, T. Qin, Q. Tao and J. Wang. Robust sparse rank learning for non-smooth ranking measures. *In Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 259-266, 2009.

[32] J. Xu and H. Li. A boosting algorithm for information retrieval. *Proceedings of the Learning to Rank workshop in the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391-398, 2007.

[33] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in learning to rank. *In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 107-114, 2008.

[34] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. *In Proceedings of the 30th annual international ACM SIGIR*

conference on Research and development in information retrieval, pages 271-278, 2007.